Limits of Breach-Resistant and Snapshot-Oblivious RAMs

Still $\Omega(b/w \cdot \log(nb/c))$

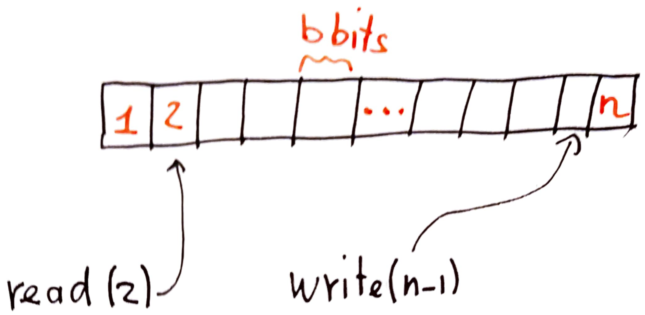
Giuseppe Persiano

Università di Salerno and Google LLC

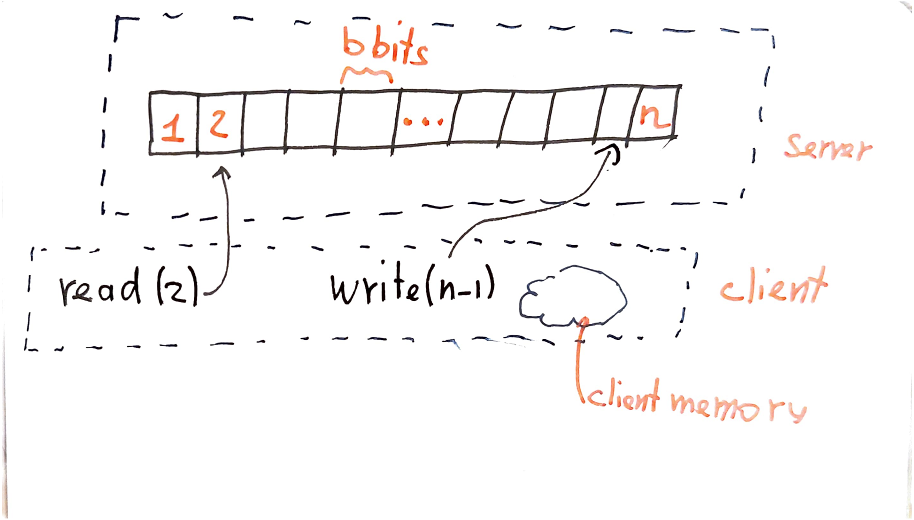
ESSA 3 in Bertinoro

Joint work with Kevin Yeo (Google LLC and Columbia U.)
Accepted to CRYPTO 23

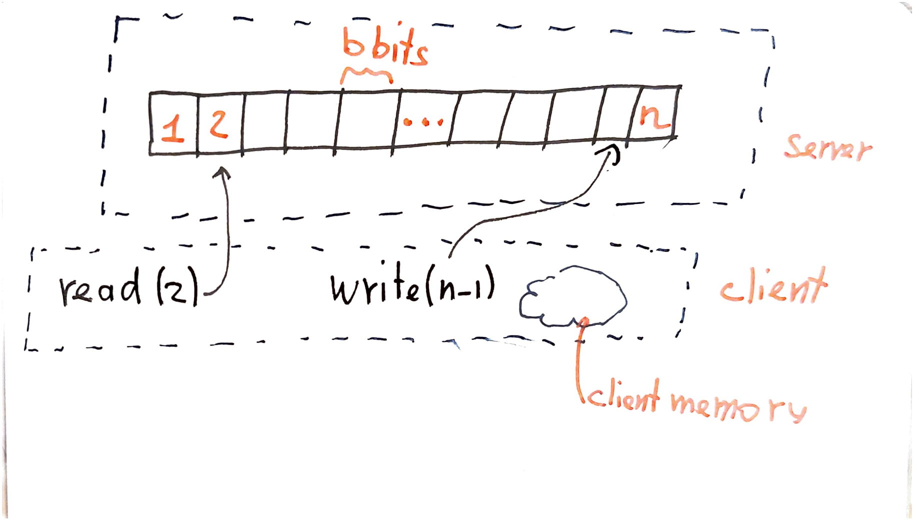
RAMS



RAMS



RAMS



Server memory words of length $w < b$

Oblivious RAMS

Hiding the access pattern to the RAM from the server

Upper bounds

- Goldreich Ostrovsky – Late 80's early 90'
  - Slowdown $O(\log^3 n)$
- ....
- Patel Persiano Raykova Yeo – 2018
  - Slowdown $O(\log n \log \log n)$
- Asharov Komogordsky Lin Peserico Shi – 2018
  - Slowdown $O(\log n)$

Oblivious RAMS

Hiding the access pattern to the RAM from the server

Upper bounds

- Goldreich Ostrovsky – Late 80's early 90'
  - Slowdown $O(\log^3 n)$
- ....
- Patel Persiano Raykova Yeo – 2018
  - Slowdown $O(\log n \log\log n)$
- Asharov Komogordsky Lin Peserico Shi – 2018
  - Slowdown $O(\log n)$

Lower bounds

- Larsen Nielsen – 2018
  - Slowdown $\Omega(\log n)$
- It holds also for Differential Privacy, some leakage

The snapshot adversary

the Server is the adversary

Snapshot Adversary

Du, Genkin, Grubbs, 2022

- The adversary gets control of the Server for L consecutive operations
  - Slowdown $O(\log L)$

The snapshot adversary

the Server is the adversary

Snapshot Adversary

Du, Genkin, Grubbs, 2022

- The adversary gets control of the Server for L consecutive operations
  - Slowdown $O(\log L)$

What if the adversary is active for more than one *window*?

Snapshot Resistant Stacks

- **read – write**

Snapshot Resistant Stacks

- **push – pop**

Snapshot Resistant Stacks

- **push – pop**
- want to hide sequence of operations
  - hide if **push** or **pop**
  - hide input to **push**

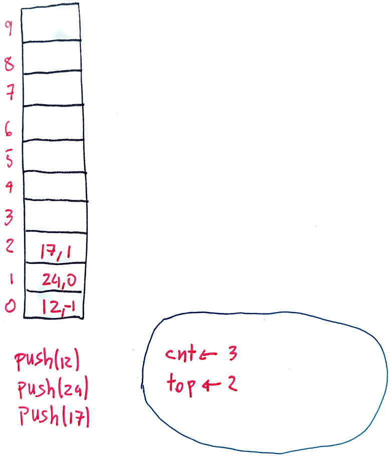
Snapshot Resistant Stacks

- **push – pop**
- want to hide sequence of operations
  - hide if **push** or **pop**
  - hide input to **push**
- hiding from whom
  - Adversary that can see S memory snapshots
  - Adversary that can see T operations transcripts

$(\infty, 0)$-snapshot secure stacks

Adversary gets snapshots of memory after all operations and no transcript

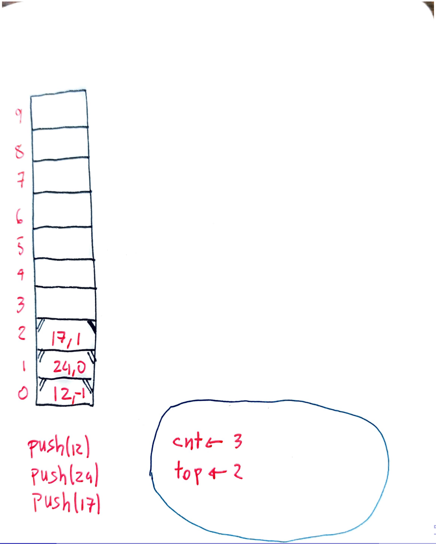
$S = \infty$ $T = 0$



$(\infty, 0)$-snapshot secure stacks

Adversary gets snapshots of memory after all operations and no transcript

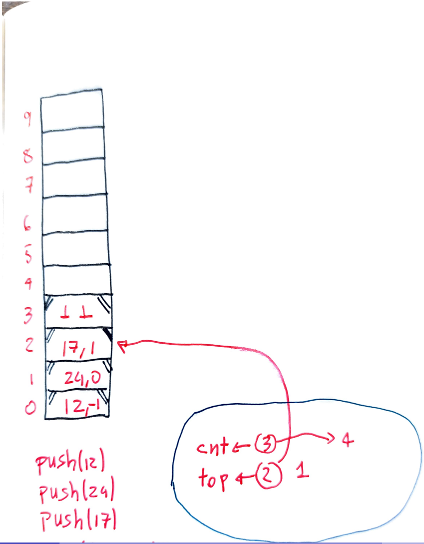
$S = \infty$ $T = 0$



$(\infty, 0)$-snapshot secure stacks

Adversary gets snapshots of memory after all operations and no transcript

$S = \infty$ $T = 0$



Snapshot Secure Stacks

- Init()
  - randomly choose encryption key K
  - set $\texttt{cnt} = 0$ and $\texttt{top} = -1$.
- Push(v)
  - upload $\mathsf{Enc}(K, (v, \texttt{top}))$ to location $\texttt{cnt}$
  - set $\texttt{top} \leftarrow \texttt{cnt}$
  - set $\texttt{cnt} \leftarrow \texttt{cnt} + 1$
- Pop()
  - download pair (v, t) from location $\texttt{top}$
  - upload a dummy encryption to location $\texttt{cnt}$
  - set $\texttt{top} \leftarrow t$
  - set $\texttt{cnt} \leftarrow \texttt{cnt} + 1$
  - return v

Snapshot Security

Snapshots:
Only difference between operation i and operation $i+1$ in location i

Snapshot Security

Snapshots:
Only difference between operation i and operation $i+1$ in location i independent from history of ops

Snapshot Security

Snapshots:
Only difference between operation i and operation $i+1$ in location i independent from history of ops

Transcripts:
Client reaches for the current `top`

Snapshot Security

Snapshots:

Only difference between operation i and operation $i+1$ in location i independent from history of ops

Transcripts:

Client reaches for the current `top`

Gives information about number of `push` *ops vs number of pop ops*

Snapshot Security

Snapshots:
Only difference between operation i and operation $i+1$ in location i independent from history of ops

Transcripts:
Client reaches for the current `top`

Gives information about number of `push` *ops vs number of pop ops*

randomly select a PRP F and write new pair at $F(\texttt{cnt})$

$(\infty, 1)$-snapshot secure stacks

$\mathcal{A}$ gets snapshots of memory after every operations and transcript for one.

$(\infty, 1)$-snapshot secure stacks

$\mathcal{A}$ gets snapshots of memory after every operations and transcript for one.

Snapshot Secure Stacks

- Init()
  - randomly choose seed S
  - randomly choose encryption key K
  - set $\texttt{cnt} = 0$ and $\texttt{top} = -1$.
- Push(v)
  - download from location $F(S, \texttt{top})$ and discard
  - upload $\mathsf{Enc}(K, (v, \texttt{top}))$ to location $F(S, \texttt{cnt})$
  - $\texttt{top} \leftarrow \texttt{cnt}$
  - $\texttt{cnt} \leftarrow \texttt{cnt} + 1$
- Pop()
  - download pair (v, t) from location $F(S, \texttt{top})$
  - upload dummy encryption at location $F(S, \texttt{cnt})$
  - set $\texttt{top} \leftarrow t$
  - set $\texttt{cnt} \leftarrow \texttt{cnt} + 1$
  - return v

This looks very promising

- Constant slowdown against *snapshot* adversary
  - *for the same price I can throw in one transcript of your choice*
- For *persistent* adversaries, stack is as hard as ORAM
  - Oblivious stack requires

$$\Omega(b/w) log(nb/c)$$

Riko Jacob, Kasper Green Larsen, Jesper Buus Nielsen. SODA 2019.

This looks very promising

- Constant slowdown against *snapshot* adversary
  - *for the same price I can throw in one transcript of your choice*
- For *persistent* adversaries, stack is as hard as ORAM
  - Oblivious stack requires

$$\Omega(b/w) log(nb/c)$$

Riko Jacob, Kasper Green Larsen, Jesper Buus Nielsen. SODA 2019.

- Actually, not! for ORAM is still

$$\Omega(b/w) log(nb/c)$$

This looks very promising

- Constant slowdown against *snapshot* adversary
  - *for the same price I can throw in one transcript of your choice*
- For *persistent* adversaries, stack is as hard as ORAM
  - Oblivious stack requires

$$\Omega(b/w) log(nb/c)$$

Riko Jacob, Kasper Green Larsen, Jesper Buus Nielsen. SODA 2019.

- Actually, not! for ORAM is still

$$\Omega(b/w) log(nb/c)$$

Sorry

The snapshot adversary

Snapshot window (t, ℓ)

- A *snapshot window* of length ℓ starting at time t.
- The adversary receives
  - snapshot of server memory content before operation t has been executed
  - transcript of server's operations for the following ℓ operations that take place at times $t, t+1, \ldots, t+\ell-1$.
- For $\ell = 0$, only memory content before operation t.

A (S, L)-snapshot adversary

Specifies a sequence of *snaspshot windows* $\mathcal{S} = ((t_1, \ell_1), \ldots, (t_s, \ell_s))$ such that

- $s \leq S$, at most S windows,
  - at most S snapshots
- $\sum \ell_i \leq L$, for a total duration of at most L operations
  - at most L transcripts

The Lower Bound

Theorem

For any $0 \leq \epsilon \leq 1/16$*, let* DS *be a* $(3, 1, \epsilon)$*-snapshot private RAM data structure for* n *entries each of* b *bits implemented over* $w = \Omega(\log n)$ *bits using client storage of* c *bits in the cell probe model. If* DS *has amortized write time* t_w *and expected amortized read time* t_r *with failure probability at most* $1/3$*, then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

The Lower Bound

Theorem

For any $0 \le \epsilon \le 1/16$, *let* DS *be a* $(3, 1, \epsilon)$*-snapshot private RAM data structure for* n *entries each of* b *bits implemented over* $w = \Omega(\log n)$ *bits using client storage of* c *bits in the cell probe model. If* DS *has amortized write time* t_w *and expected amortized read time* t_r *with failure probability at most* $1/3$, *then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

n *logical* blocks of b bits

The Lower Bound

Theorem

For any $0 \leq \epsilon \leq 1/16$, let DS be a $(3, 1, \epsilon)$-snapshot private RAM data structure for n entries each of b bits implemented over $w = \Omega(\log n)$ bits using client storage of c bits in the cell probe model. If DS has amortized write time t_w and expected amortized read time t_r with failure probability at most $1/3$, then

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

$w < b$ is size *physical* words

The Lower Bound

Theorem

For any $0 \leq \epsilon \leq 1/16$, *let* DS *be a* $(3, 1, \epsilon)$*-snapshot private RAM data structure for* n *entries each of* b *bits implemented over* $w = \Omega(\log n)$ *bits using client storage of* c *bits in the cell probe model. If* DS *has amortized write time* t_w *and expected amortized read time* t_r *with failure probability at most* $1/3$, *then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

Client has c bits of *local memory*

The Lower Bound

Theorem

For any $0 \leq \epsilon \leq 1/16$*, let* DS *be a* $(3, 1, \epsilon)$*-snapshot private RAM data structure for* n *entries each of* b *bits implemented over* $w = \Omega(\log n)$ *bits using client storage of* c *bits in the cell probe model. If* DS *has amortized write time* t_w *and expected amortized read time* t_r *with failure probability at most* $1/3$*, then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

Adversary receives at most 3 memory *snapshots* and 1 operation *transcript*

The Lower Bound

Theorem

For any $0 \leq \epsilon \leq 1/16$, *let* DS *be a* $(3, 1, \epsilon)$*-snapshot private RAM data structure for* n *entries each of* b *bits implemented over* $w = \Omega(\log n)$ *bits using client storage of* c *bits in the cell probe model. If* DS *has amortized write time* t_w *and expected amortized read time* t_r *with failure probability at most* $1/3$, *then*

$$t_r + t_w = \Omega\left(b/w \cdot \log(nb/c)\right).$$

ϵ is the adversary's advantage in the security game

The security game

$\mathsf{Expt}^{n,\beta}_{\mathsf{DS},\mathcal{A}}$

- Receive $(O_0, O_1, ((t_1, \ell_1), \ldots, (t_s, \ell_s)))$ from $\mathcal{A}_0(1^n)$.
- Set $\mathcal{L} \leftarrow \emptyset$, $\mathsf{DS} \leftarrow (R_1, \ldots, R_n)$, $i \leftarrow 1$.
- While $i \leq |O_\beta|$:
  - If $i = t_j$ for some $1 \leq j \leq s$:
    - Set $\mathcal{L} \leftarrow \mathcal{L} \,||\, (\text{memory}, M)$.
    - For $k = 1, \ldots, \ell_j$:
      Execute $\mathsf{DS}^{\mathsf{LRead},\mathsf{LWrite}}(O_b[i])$ and set $i \leftarrow i + 1$.
  - Else:
    Execute $\mathsf{DS}^{\mathsf{Read},\mathsf{Write}}(O_b[i])$ and set $i \leftarrow i + 1$.
- Return $\mathcal{A}_1(\mathcal{L})$.

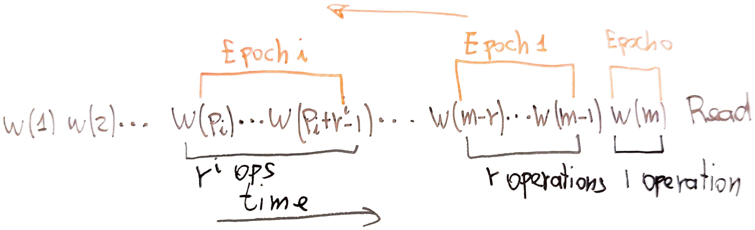
$$\left|\Pr[\mathsf{Expt}^{n,0}_{\mathsf{DS},\mathcal{A}} = 1] - \Pr[\mathsf{Expt}^{n,1}_{\mathsf{DS},\mathcal{A}} = 1]\right| \leq \epsilon,$$

for all PPT $\mathcal{A}$ that are (S, L)-snapshot adversaries.

The Epoch structure

The sequence and the epochs

- n *logical indices*
- $m \leftarrow \{n/2+1, \ldots, n\}$
- m writes of **random** b-bit blocks at indices $1, 2, \ldots, m$
- followed by one read.



A $(3,1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1

A $(3, 1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices

A $(3, 1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1

A $(3, 1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch

A $(3,1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch
- **correctness of** O_1
  touch about b/w cells updated in epoch i

A $(3,1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch
- **correctness of** O_1
  touch about b/w cells updated in epoch i
  - epochs preceding epoch i are **independent**

A $(3, 1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch
- **correctness of** O_1
  touch about b/w cells updated in epoch i
  - epochs preceding epoch i are **independent**
  - epochs following epoch i are **not large enough**

A $(3, 1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch
- **correctness of O_1**
  touch about b/w cells updated in epoch i
  - epochs preceding epoch i are **independent**
  - epochs following epoch i are **not large enough**
  - pick i so that client memory is **too small**

A $(3,1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch
- **correctness of O_1**
  touch about b/w cells updated in epoch i
  - epochs preceding epoch i are **independent**
  - epochs following epoch i are **not large enough**
  - pick i so that client memory is **too small**
- **correctness of O_0**
  read of O_0 does not depend on epoch i

A $(3, 1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch
- **correctness of** O_1
  touch about b/w cells updated in epoch i
  - epochs preceding epoch i are **independent**
  - epochs following epoch i are **not large enough**
  - pick i so that client memory is **too small**
- **correctness of** O_0
  read of O_0 does not depend on epoch i
- **security**
  but if it does not, then security fails

A $(3,1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch
- **correctness of O_1**
  touch about b/w cells updated in epoch i
  - epochs preceding epoch i are **independent**
  - epochs following epoch i are **not large enough**
  - pick i so that client memory is **too small**
- **correctness of O_0**
  read of O_0 does not depend on epoch i
- **security**
  but if it does not, then security fails
- **final step**
  this holds for *all* epochs except for those that have fewer than c/b writes.

A $(3,1)$-snapshot adversary – Intuition

- Two sequences of operations O_0, O_1
  - Both write **random** blocks to the first m indices
  - O_0 reads index 1
  - O_1 reads a randomly selected index j written in the i-th epoch
- **correctness of O_1**
  touch about b/w cells updated in epoch i
  - epochs preceding epoch i are **independent**
  - epochs following epoch i are **not large enough**
  - pick i so that client memory is **too small**
- **correctness of O_0**
  read of O_0 does not depend on epoch i
- **security**
  but if it does not, then security fails
- **final step**
  this holds for *all* epochs except for those that have fewer than c/b writes.
- **we have a lower bound** $\Omega(b/w \cdot \log(nb/c))$

A $(3,1)$-snapshot adversary – Part 0

$\mathcal{A}_0^i(1^n)$

- Randomly select integer m from $[n/2, n]$.
- Randomly and ind. select $\mathsf{B}_1, \ldots, \mathsf{B}_m \leftarrow \{0,1\}^b$.
- Set $O_0 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(m))$.
- Randomly select $j \in [p_i, p_i + r^i - 1]$,
- Set $O_1 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(j))$.
- Set $\mathcal{S} = ((p_i, 0), (p_i + r^i, 0), (m+1, 1))$.
- Return (O_0, O_1, S).

A $(3, 1)$-snapshot adversary – Part 0

$\mathcal{A}_0^i(1^n)$

- Randomly select integer m from $[n/2, n]$.
- Randomly and ind. select $\mathsf{B}_1, \ldots, \mathsf{B}_m \leftarrow \{0,1\}^b$.
- Set $O_0 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(m))$.
- Randomly select $j \in [p_i, p_i + r^i - 1]$,
- Set $O_1 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(j))$.
- Set $\mathcal{S} = ((p_i, 0), (p_i + r^i, 0), (m + 1, 1))$.
- Return (O_0, O_1, S).

- $(p_i, 0)$: snapshot of server memory before epoch i

A $(3,1)$-snapshot adversary – Part 0

$\mathcal{A}_0^i(1^n)$

- Randomly select integer m from $[n/2, n]$.
- Randomly and ind. select $\mathsf{B}_1, \ldots, \mathsf{B}_m \leftarrow \{0,1\}^b$.
- Set $O_0 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(m))$.
- Randomly select $j \in [p_i, p_i + r^i - 1]$,
- Set $O_1 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(j))$.
- Set $\mathcal{S} = ((p_i, 0), (p_i + r^i, 0), (m+1, 1))$.
- Return (O_0, O_1, S).

- $(p_i, 0)$: snapshot of server memory before epoch i
- $(p_i + r^i, 0)$: snapshot of server memory after epoch i

A $(3, 1)$-snapshot adversary – Part 0

$\mathcal{A}_0^i(1^n)$

- Randomly select integer m from $[n/2, n]$.
- Randomly and ind. select $\mathsf{B}_1, \ldots, \mathsf{B}_m \leftarrow \{0, 1\}^b$.
- Set $O_0 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(m))$.
- Randomly select $j \in [p_i, p_i + r^i - 1]$,
- Set $O_1 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(j))$.
- Set $\mathcal{S} = ((p_i, 0), (p_i + r^i, 0), (m + 1, 1))$.
- Return (O_0, O_1, S).

- $(p_i, 0)$: snapshot of server memory before epoch i
- $(p_i + r^i, 0)$: snapshot of server memory after epoch i
- $(m + 1, 1)$: snapshot before read and transcript of read operation

A $(3, 1)$-snapshot adversary – Part 0

$\mathcal{A}_0^i(1^n)$

- Randomly select integer m from $[n/2, n]$.
- Randomly and ind. select $\mathsf{B}_1, \ldots, \mathsf{B}_m \leftarrow \{0, 1\}^b$. **Important**
- Set $O_0 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(m))$.
- Randomly select $j \in [p_i, p_i + r^i - 1]$,
- Set $O_1 = (\mathsf{write}(1, \mathsf{B}_1), \ldots, \mathsf{write}(m, \mathsf{B}_m), \mathsf{read}(j))$.
- Set $\mathcal{S} = ((p_i, 0), (p_i + r^i, 0), (m + 1, 1))$.
- Return (O_0, O_1, S).

- $(p_i, 0)$: snapshot of server memory before epoch i
- $(p_i + r^i, 0)$: snapshot of server memory after epoch i
- $(m + 1, 1)$: snapshot before read and transcript of read operation

A $(3,1)$-snapshot adversary – Part 1

- U_i memory locations overwritten during epoch i
  - by comparing the **initial** and **final** snapshot of epoch i
- V_i memory locations overwritten since epoch i
  - by comparing the **final** snapshot of epoch i with snapshot **before the read**
- W_i memory location overwritten during epoch i that have not been modified when the read starts
  - $W_i = U_i \setminus V_i$
- Q_j cells from W_i read during $\mathsf{read}(j)$,
- $|Q_j| \approx b/w$
  - $\mathcal{A}^1$ returns 0 iff $|Q_j| \le \rho \cdot b/w$

The coding argument

Suppose

$$t_w = o(b/w \log(nb/c))$$

then there exists $\rho > 0$ such that, for most epochs i,

$$|Q_j| \geq \rho \cdot b/w$$

with probability $\geq 1/8$ for j in epoch i.

The coding argument

Suppose

$$t_w = o(b/w \log(nb/c))$$

then there exists $\rho > 0$ such that, for most epochs i,

$$|Q_j| \geq \rho \cdot b/w$$

with probability $\geq 1/8$ for j in epoch i.

Suppose not.

The coding argument

Suppose

$$t_w = o(b/w \log(nb/c))$$

then there exists $\rho > 0$ such that, for most epochs i,

$$|Q_j| \geq \rho \cdot b/w$$

with probability $\geq 1/8$ for j in epoch i.

Suppose not.
Then we can encode the $r^i \cdot b$ bits of epoch i using fewer bits.

The coding argument - I

A coding game

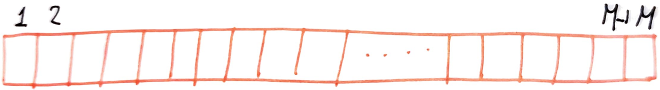
- S wants to send B^i to R
  - the r^i blocks from epoch i
- S and R share
  - B^{-i} (all except epoch i)
  - randomness $\mathcal{R}$ to execute DS.

$$\mathcal{H}(\mathsf{B}^i|\mathcal{R}, \mathsf{B}^{-i}) = r^i \cdot b.$$

The coding argument - II

- S and R execute all epochs $> i$

$$\mathrm{write}(1, \mathrm{B}_1), \ldots, \mathrm{write}(p_i - 1, \mathrm{B}_{p_i - 1})$$

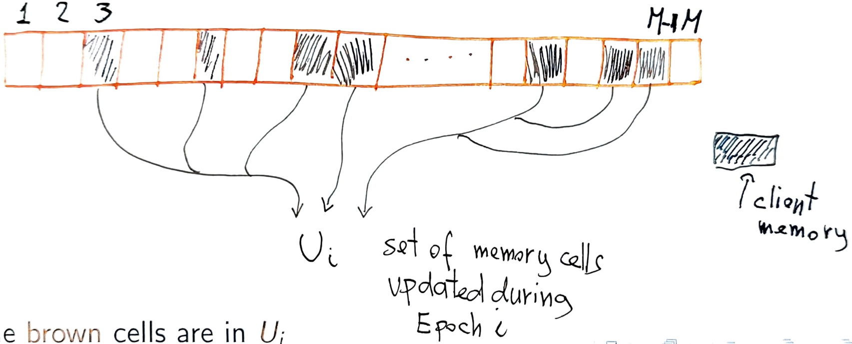


The coding argument

- S executes epoch i

$$\text{write}(p_i, \mathrm{B}_{p_i}), \ldots, \text{write}(p_i + r^i - 1, \mathrm{B}_{p_i + r^i - 1})$$

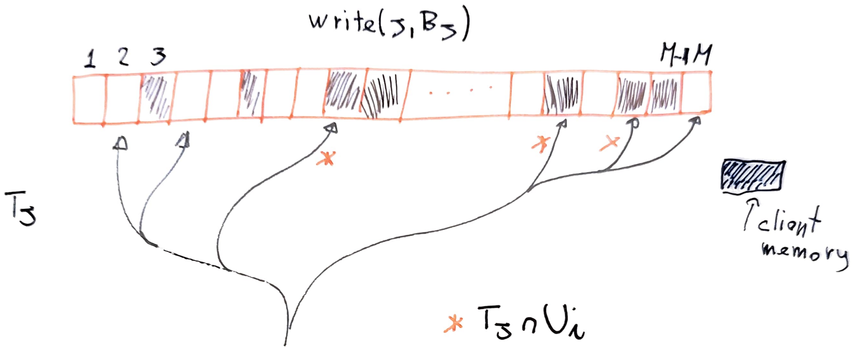
- Note: R cannot execute epoch i



The brown cells are in U_i

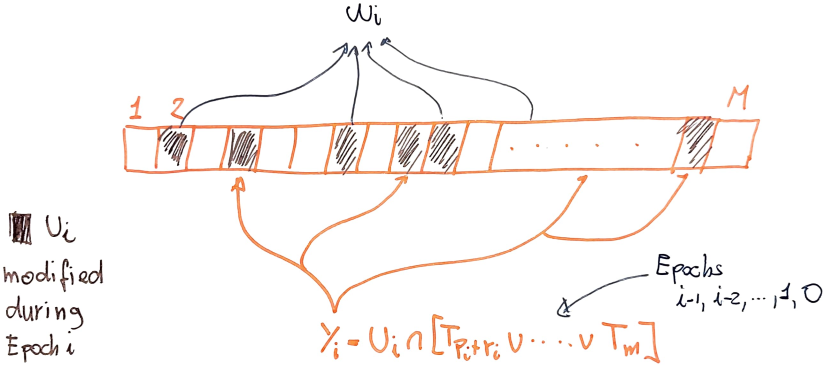
The coding argument

- S and R execute epochs $< i$
  - R needs some help
    - client memory: c bits.
- For $j = p_{i-1}, \ldots, m$
  - execute write(j, B_j) touching T_j
  - R needs $U_i \cap T_j$ (cell location and content)



The coding argument

- c bits + set $Y_i := U_i \cap (T_{p_i+r_i} \cup \cdots \cup T_m)$



The coding argument

$\mathbb{S}$ memory state after write(m, B_m)

- For $j = p_i, \ldots, p_i + r^i - 1$
  - S and R execute read(j) starting from $\mathbb{S}$
  - R needs $Q_j := W_i \cap T_j^m$
  - if read errs or $Q_j > \rho b/w$
    - B_j is added to encoding
  - else
    - Q_j is added to encoding

Length of encoding

Length depends on

- Set Y_i
  - for most epochs i, $\mathbb{E}[|Y_i|] \leq r^{i-1} b/w$
- Set Q_j
  - By assumption $|Q_j| < \rho \cdot b/w$ with prob $\geq 7/8$

Length of encoding

Length depends on

- Set Y_i
  - for most epochs i, $\mathbb{E}[|Y_i|] \leq r^{i-1} b/w$
- Set Q_j
  - By assumption $|Q_j| < \rho \cdot b/w$ with prob $\geq 7/8$

Encoding is too small

Getting there...

$$t_w = o(b/w \log(nb/c))$$

implies that, for most epochs i,

$$|Q_j| \geq \rho \cdot b/w$$

with probability $\geq 1/8$ for j in epoch i. from epoch i.

Getting there...

$$t_w = o(b/w \log(nb/c))$$

implies that, for most epochs i,

$\mathcal{A}$ outputs 1 with probability $\geq 1/8$ when reading j from epoch i.

Getting there...

$$t_w = o(b/w \log(nb/c))$$

implies that, for most epochs i,

$\mathcal{A}$ outputs 1 with probability $\geq 1/8$ when reading j from epoch i.

If $\epsilon = 1/16$ then $\mathcal{A}$ outputs 1 with probability $\geq 1/16$ when reading m

Getting there...

$$t_w = o(b/w \log(nb/c))$$

implies that, for most epochs i,

$\mathcal{A}$ outputs 1 with probability $\geq 1/8$ when reading j from epoch i.

If $\epsilon = 1/16$ then $\mathcal{A}$ outputs 1 with probability $\geq 1/16$ when reading m

$\text{read}(m)$ must touch $\geq \rho \cdot b/w$ cells from epoch i

Getting there...

$$t_w = o(b/w \log(nb/c))$$

implies that, for most epochs i,

$\mathcal{A}$ outputs 1 with probability $\geq 1/8$ when reading j from epoch i.

If $\epsilon = 1/16$ then $\mathcal{A}$ outputs 1 with probability $\geq 1/16$ when reading m

$\mathsf{read}(m)$ must touch $\geq \rho \cdot b/w$ cells from epoch i

$$\Omega(b/w \cdot \log nb/c)$$

Wrapping up

Now...

If writes are fast

$$t_w = o(b/w \log(nb/c))$$

then read(j) in epoch i has $Q_j = \Omega(b/w)$ with prob at least $1/8$.

Wrapping up

Now...

If writes are fast

$$t_w = o(b/w \log(nb/c))$$

then read(j) in epoch i has $Q_j = \Omega(b/w)$ with prob at least 1/8.

Reading 1

Must touch from each large epoch $O(b/w)$ cells otherwise we lose security.

$$\Omega(b/w \cdot \log(nb/c))$$